MyGa : manuel d'utilisation

Adel BEN HAJ YEDDER.

19 novembre 2002

1 Introduction

Ce programme est une implémentation d'un algorithme génétique avec codage réel écrit en Fortran. Les types d'opérateurs développés sont très limités et seuls certains des opérateurs connus pour leur efficacité ont été implémenté. Le but de ce programme, lorsqu'il a été développé, était d'avoir un algorithme génétique en fortran facile à utiliser et efficace sur le problème pour lequel il a été écrit.

Le programme est distribué afin de fournir aux programmeurs fortran un outil pour tester et/ou utiliser une optimisation par les algorithmes génétiques sur le propre problème.

Ce programme est fourni sans aucune garantie ni aucun support autre que ce document. Si vous l'utilisez vous êtes le SEUL responsable de son utilisation et en particulier des résultats qu'il donne et de ce que vous en faite ensuite.

2 Installation

• **Système d'exploitation :** Le programme a été écrit et testé pour ***Linux***. Cependant il devrait marcher sur les autres systèmes du type *N*X moyennant de légères modifications.
• **Compilateur :** Le programme est écrit en ***Fortran 77***. Vous devez donc utiliser un compilateur supportant le Fortran 77. Pour définir le compilateur éditez le fichier **Makefile** et modifiez la première ligne. Le compilateur par défaut est `g77`.
• **Extraction des fichiers et compilation :** Tapez les lignes de commandes suivantes :

```
> gunzip MyGa_1.0.tar.gz
> tar xvf MyGa_1.0.tar
> cd MyGa_1.0
> make
```

3 Utilisation

L'optimisation dans MyGa se fait en ***minimisant*** une fonction $f(x)$ calculée dans **func.f** avec `x` un vecteur de dimension `dimx` définie dans **parameter.inc**.

3.1 Exemples

Le fichier **func.f** fourni contient 5 exemples de fonctions tests. Le premier exemple est celui de la fonction sphère en dimension $dimx = 10$. Pour tester cet exemple tapez :

```
> make
> MyGa
```

L'évolution de la meilleur solution au cours des générations est à la fois affichée à l'écran et sauvegardée dans les fichiers **bestfx.dat** (la valeur de la fonction) et **bestx.dat**. Le fichier **resultat.dat** contient le résultat final obtenu. Pour les tester les autres exemples il suffit d'éditer le fichier **func.f** et de choisir un exemple en changeant la ligne `goto 10`

en `goto` *num* où $num \in \{10, 20, 30, 40, 50\}$. Editez ensuite le fichier **parameter.inc** pour choisir le dimension `dimx` en fonction du cas choisi. Vous pouvez laisser la dimension inchangée sauf pour le cas $num = 40$ où la dimension doit être égale à 2. Enfin éditez le fichier **param.dat** pour changer les deux lignes

```
xmin=10*-1.D1,
xmax=10*1.D1,
```

pour préciser les bornes dans lesquels s'effectuera la recherche. Remplacez `10` par la valeur de `dimx` dans les deux lignes et remplacez `-1.D1` et `1.D1` respectivement par la borne inférieure et la borne supérieure des intervalles de recherche. Recompilez et relancez le programme. D'une façon générale pour effectuer une recherche dans

$$E = [xmin_1, xmax_1] \times \cdots \times [xmin_{dimx}, xmax_{dimx}]$$

les bornes doivent s'écrire sous la forme :

xmin=$xmin_1$,$\cdots$,$xmin_{dimx}$,
xmax=$xmax_1$,$\cdots$,$xmax_{dimx}$,

3.2 Utilisation simple

Pour utiliser très rapidement MyGa pour minimisez une autre fonction procédez de la façon suivante :

1. Changez dans le fichier **parameter.inc** la dimension de l'espace de recherche `dimx`. C'est le nombre de vos paramètres à optimisez.
2. Ecrivez dans le fichier **func.f** votre fonction à minimiser. Cette fonction doit recevoir un vecteur `x` de taille `dimx` et renvoyer la valeur de la fonction `fx`.
   Si votre fonction correspond à un programme que vous utilisez avec vos propres notations et noms de variables, le mieux est de "ranger" vos variables à optimiser dans le vecteur `x`. Ainsi, vous gardez vos propres noms de variables et la routine **func.f** ne "voit" que le vecteur `x` et la valeur `fx`. De même si vous devez transmettre des paramètres à votre fonction ou que vous devez faire des initialisations vous devez effectuer ces opérations sans modifier la structure de la routine **func.f**.
3. Changez dans le fichier **param.dat** les bornes de l'espace de recherche comme indiqué dans la section 3.1.
4. Compilez avec `make` et exécutez le programme.

3.3 Utilisation avancée

On suppose maintenant que les étapes de la section 3.2 ont été réalisées avec succès. Dans cette section on va parcourir les deux fichiers **parameter.inc** et **param.dat**.

3.3.1 Le fichier parameter.inc

Dans ce fichier il y a 4 paramètres à régler :
• `popsize :` La taille de la population de chaque générations. Ce programme a donné les meilleurs performances avec un taille égale à 10.
• `dimx :` La dimension de l'espace de recherche.
• `genemax :` Le nombre maximum de générations avant l'arrêt du programme.
• `genemaxmut :` C'est un paramètre qui règle la décroissance de la force de mutation du type `nonun2` définie dans la section 3.3.2. Ce paramètre doit être `genemaxmut`≥`genemax`. Pour une décroissance plus lente de la force de mutation essayez avec `genemaxmut=1.5*genemax` par exemple.

3.3.2 Le fichier param.dat

- **Affichage :** Le paramètre `affichecran` peut prendre les valeurs 0, 1, 2 et 3. A chaque niveau des informations supplémentaires sont ajoutées.
  - 0 : aucun affichage
  - 1 : affichage de la dernière génération et de la meilleur solution.
  - 2 : on affiche la valeur de la meilleur solution trouvée à chaque génération.
  - 3 : la génération initiale est affichée.
- **Sauvegarde :** Le paramètre `savefich` peut prendre les valeurs 0, 1 et 2.
  - 0 : aucune sauvegarde. Sans intérêt si en plus vous utilisez `affichecran=0` !
  - 1 : sauvegarde de la meilleur solution (fichier **resultat.dat**) et de l'état des calculs (fichier **restart.dat**) pour une reprise éventuelle des calculs.
  - 2 : sauvegarde de la meilleur solution à chaque génération (fichier **bestx.dat**) et de la valeur de la fonction correspondante (fichier **bestfx.dat**).
- **Tests d'arrêt :** Deux types de tests d'arrêt sont possibles. Quand vous voulez faire des tests le programme peut s'arrêter soit lorsque la solution (`solx`) est atteinte soit lorsque la valeur du minimum de f (`solfx`) est atteinte. Dans le cas général le programme s'arrête quand le nombre d'itérations maximal (`restartfin`) est atteint.
  - Solution atteinte : mettre le paramètre `typeconv='solx',`, `solx=`$x_1, \cdots, x_{dimx}$`,` et `eps=`ϵ`,` où $(x_1, \cdots, x_{dimx})$ sont les coordonnées de la solution et où ϵ définit la précision.
  - Valeur du minimum atteinte : mettre le paramètre `typeconv='solfx',`, `solfx=minimum,` et `eps=`ϵ`,` où `minimum` est la valeur du minimum de la fonction f.
  - Cas général : mettre le paramètre `typeconv='',` et `restartfin=iterfin,` où `iterfin` est la génération à laquelle le programme doit s'arrêter qui doit être `iterfin` $\leq$ `genemax`.
    Plus précisément `restartini=iterdebut,` et `restartfin=iterfin,` définissent la génération du début et la génération de la fin du programme. Pour un premier calcul on doit avoir `restartini=1,`. Dans ce cas la population initiale est générée aléatoirement dans l'espace de recherche $E = [xmin_1, xmax_1] \times \cdots \times [xmin_{dimx}, xmax_{dimx}]$.
    Pour reprendre un ancien calcul arrêté à une génération $gen_1 < genemax$ utilisez `restartini=`gen_1`,` et `restartfin=iterfin,` avec $gen_1 < iterfin \leq genemax$. Dans ce cas le calcul sera repris avec la population qui sera relue dans le fichier **restart.dat** crée après le premier calcul. On note qu'un calcul fait en deux temps n'est pas toujours le même qu'un calcul fait en une seule étape vu que le fichier **restart.dat** contient uniquement la population à laquelle le calcul doit reprendre mais ne contient pas d'autres informations. Ce détail n'a pas beaucoup d'importance sauf dans le cas où on fait des tests de performances ou où on cherche a reproduire parfaitement un ancien calcul.
- **Initialisation de la population :** La population initiale est générée aléatoirement dans le cas général. Quand le paramètre `initpop=m,` avec `0<m`$\leq$`dimx`, le programme va lire `m` individus dans le fichier **initx.dat** qui vont remplacer `m` individus de la population aléatoire. Dans ce cas vous devez introduire ces `m` individus dans le fichier **initx.dat** en y ajoutant une coordonnée par ligne. Ceci peut être utile si vous introduire des individus qui vous semblent proches de la solution. Mais attention à la convergence prématurée qui peut être provoquée par un individus très bons (celui que vous ajoutez) par rapport aux autres.
- **Espace de recherche :** Comme indiqué dans la section 3.1 l'espace de recherche est défini par les bornes `xmin=`$xmin_1, \cdots, xmin_{dimx}$`,` et `xmax=`$xmax_1, \cdots, xmax_{dimx}$`,`. Le programme ne prend en compte aucun autre type de contraintes. Une façon simple de les prendre en compte est de pénaliser fortement une violation des contraintes dans la fonction coût f. C'est loin d'être la meilleur façon de prendre en compte les contraintes mais, faute de mieux, c'est une façon qui marche dans beaucoup de cas de contraintes simples.

• **Croisement :** Les paramètres correspondant à l'opérateur de croisement sont `pc`, `typec`, `alpha`, `nbmultip` et `tm`.
La probabilité d'appliquer un croisement à un parent est donnée par `pc`.
Quatre types de croisement sont possibles :

- `typec='bary',` : croisement barycentrique avec le paramètre `alpha=`α`,` définissant les poids α et $(1-\alpha)$.
- `typec='baryal',` : croisement barycentrique où le coefficient α est choisi aléatoirement dans l'intervalle $[0,1]$.
- `typec='multip',` : croisement multi-points avec le nombre de points de coupure donné par `nbmultip=`m`,` où `m` est un entier `0<m<dimx`.
- `typec='sbx',` : croisement dit SBX avec le paramètre `alpha=`α`,` définissant la forme de la loi de probabilité utilisée. Prendre $\alpha = 1$ ou 2 par exemple.

Dans le cas où les deux parents sont très proches, la probabilité de leur mutation (voir partie mutation ci-dessous) est augmenté et devient égale à `tm`. Le but est de muter les deux enfants générés qui ressemblent beaucoup à leur parents.
• **Mutation :** Les paramètres de cet opérateur sont `pm`, `typem`, `b`, `nbstag` et `kb`.
La probabilité d'appliquer une mutation à un individu est donnée par `pm`.
Trois types de croisement sont possibles :

- `typem='varnor',` : mutation gaussienne avec un écart type σ qui décroît vers zéro au cours des générations.
- `typem='nonuni',` : mutation non-uniforme (voir [1, (Section 1.3.2.4)] et [3] pour la définition de cette mutation). Cette mutation est réglée par le paramètre `b`. Une petite valeur de `b` (∼1.) favorise l'exploration et une valeur élevée accélère la recherche locale.
- `typem='nonun2',` : c'est une variante de la mutation non-uniforme qui souvent donne de meilleurs résultats.

Echapper aux minima locaux : dans le cas des mutations de type `nonuni` et `nonun2` les deux paramètres `nbstag` et `kb` permettent de sortir dans certains cas des minima locaux dans lesquels le programme peut se trouver piégé. En effet, si aucune amélioration de la meilleur solution n'est réalisée après `nbstag` générations, la valeur du paramètre `b` devient, à cette génération, `kb*b` avec `kb < 1`. Ceci a pour effet d'augmenter la force de mutation et favoriser une exploration de l'espace de recherche.
• **Recherche locale :** définie par les paramètres `RafLoc` et `nbtot`. Cette procédure est peu efficace et déclenche souvent une convergence prématurée. A éviter en laissant `RafLoc='NOrafloc',`.
• **Sélection :** deux types de sélections sont possibles.

- `types='roulet',` : sélection par roulette.
- `types='rs',` : sélection par roulette stochastique.

• **Elitisme :** activé si `elit=1,` et désactivé si `elit=0,`. L'élitisme consiste à assurer la décroissance de la meilleur solution à chaque génération. En pratique, si à une génération n donnée aucun individu n'est meilleur que la solution de la génération précédente $n-1$, celle-ci est transférée à la génération n.
• **Nichage :** cette technique permet de favoriser l'exploration spatiale en pénalisant les individus qui se trouvent dans la même région de l'espace. Les paramètres sont `niche`, `q`, `sigmaniche` et `alphaniche`. Voir [3] pour les détails de cette technique.

- `niche=1,` : pour activer cette technique et `0` pour la désactiver.
- `q=p,` : pour utiliser la norme $||\cdot||_p$ dans le calcul de la distance.
- `sigmaniche=`σ`,` : doit être < 1.
- `alphaaniche=`α`,` : pour régler l'intensité du nichage. Avec α petit ($\alpha < 1$) le nichage est peu intense et avec α grand le nichage est très intense.

• **Scaling :** voir [1, (Section 1.3.2.4)] et [2] pour les détails. Les paramètres sont `scaling`, `scal`, `scalingvar`, `scalvar1`, `scalvar2` et `scalvarp`. Les deux premiers paramètres pour le premier scaling (5 possibilités) et 4 derniers pour un second scaling variable (facultatif) en

fonction du temps (5 possibilités). Le scaling qui donné les meilleur résultats est le σ-scaling que l'on peut choisir par `scaling='sigma',`. Les autres scaling possibles sont `exp` (en e^x), `xexp` (en xe^x), `log` (en $\log(1+x)$) et `puiss` (en x^{scal}). Ces mêmes sacling sont adaptés pour le scaling variable.

4 Remarques

• **Initialisation du générateur de nombres aléatoires :** Actuellement le générateur de nombres aléatoires est initialisé toujours de la même façon. Par conséquent, deux exécutions successives du programme produisent le même résultat. Pour initialiser le générateur autrement éditez le fichier **MyGa.f** et changer au début du programme la ligne `irandom=-1` par `irandom=time()`.

Références

[1] A. Ben Haj Yedder. *Optimisation numérique et Contrôle optimal : (applications en chimie moléculaire).* PhD thesis, Ecole Nationale des Ponts et Chaussées, 2002.

[2] Z. Michalewicz. *Genetic algorithms + data structure = evolution programs.* Springer, 1999.

[3] Mourad Sefrioui. *Algorithmes Evolutionnaires pour le calcul scientifique. Application la mécanique des fluides et l'électromagnetisme.* PhD thesis, Université Pierre et Marie Curie, Avril 1998.